

ALLSAT compressed with wildcards.

Part 1: Converting CNF's to orthogonal DNF's

Marcel Wild

ABSTRACT: For most branching algorithms in Boolean logic “branching” means “variable-wise branching”. We present the apparently novel technique of clause-wise branching, which is used to solve the ALLSAT problem for arbitrary Boolean functions in CNF format. Specifically, it converts a CNF into an orthogonal DNF, i.e. into an exclusive sum of products. Our method is enhanced by two ingredients: The use of any good SAT-solver and wildcards beyond the common don't-care symbol.

1 Introduction

In his 1979 landmark paper [V] Leslie Valiant shifted attention from the SAT problem to the #SAT problem, i.e. to the task of calculating the exact cardinality of the model set $\text{Mod}(\varphi) \subseteq \{0, 1\}^w$ of a given Boolean function $\varphi = \varphi(x_1, \dots, x_w)$. He showed that many #SAT problems are so-called #P-hard which implies they are at least as difficult as NP-hard problems. Even problems for which SAT is trivial can be #P-hard, such as #DNFSAT. Solving #SAT e.g. has applications for approximate and probabilistic reasoning.

The ALLSAT problem for φ , our article's topic, extends #SAT in so far as not just the number $|\text{Mod}(\varphi)|$ is required but the models themselves. In the literature often the underlying Boolean function (or formula) φ is not pointed out explicitly. In fact one may be interested in a subset $\text{SpMod}(\varphi)$ of “special” models which would be hard or impossible to capture by a Boolean formula. In the last fifty years a variety of such “enumeration problems” have been considered [Was]. The motivations for doing so (e.g. optimization beyond the scope of linear programming) are laid out e.g. in Foreword 2 of [M]. We won't dwell on optimization in the sequel. Rather the present article sticks to theory (algorithm analysis), and to a strict Boolean function framework.

Since $|\text{Mod}(\varphi)|$ can be exponential in the input length, one commonly regards the ALLSAT problem as *solvable* when the enumeration of $\text{Mod}(\varphi)$ can be achieved in *total polynomial* time. It turns out that many classes \mathcal{C} of Boolean functions whose #SAT problem is #P-hard nevertheless have a solvable ALLSAT-problem, e.g. the class of all Boolean DNF's. A sufficient condition for a solvable ALLSAT problem was formulated in the somewhat hidden Fact 7 of [V]. Our Corollary 2 with its *hereditary* classes \mathcal{C} resembles¹ Fact 7. Roughly speaking Corollary 2 states that tackling the ALLSAT problem for a hereditary class \mathcal{C} scales proportional to the best SAT-solver available for \mathcal{C} . In particular, a polynomial-time SAT-solver triggers a solvable ALLSAT problem for \mathcal{C} .

¹In [R] Fact 7 has been extended to an interesting hierarchy of listing problems (e.g. so-called \mathcal{LP} -complete problems). A careful analysis of touching points with our article is pending.

Unfortunately, from a practical point of view a one-by-one enumeration of $\text{Mod}(\varphi)$, in polynomial total time or not, gets tiresome when $|\text{Mod}(\varphi)|$ goes into the trillions. How we mend this state of affairs is surveyed in parts of the remainder (mainly in 1.1) of the introduction. The detailed section break-up follows in 1.2.

1.1 Enumerating a set of objects usually means that they are listed *one-by-one*. Thus, if the Boolean function $\varphi_0 : \{0, 1\}^w \rightarrow \{0, 1\}$ has $w = 9$ and is defined by $\varphi_0(x_1, \dots, x_9) = x_2 \vee \bar{x}_6$ then enumerating the model set $\text{Mod}(\varphi_0)$ in this strict sense forces us to list 384 length 9 bitstrings. Since it is more economic to write

$$(1) \quad \text{Mod}(\varphi_0) = (2, \mathbf{1}, 2, 2, 2, \mathbf{2}, 2, 2, 2) \uplus (2, \mathbf{0}, 2, 2, 2, \mathbf{0}, 2, 2, 2),$$

we henceforth mean by an *enumeration* of $\text{Mod}(\varphi)$ a partition of $\text{Mod}(\varphi)$ into such disjoint 012-rows. In other words, solving the ALLSAT problem for φ amounts to find an orthogonal DNF of φ . (As to “orthogonal”, another term is “disjoint sum of products, see also [CH]). Here comes some handy notation right away: $\text{zeros}(r)$, $\text{ones}(r)$, and $\text{twos}(r)$ are the sets of *positions* $i \in [w] := \{1, 2, \dots, w\}$ where the 0’s, 1’s, and the don’t-care symbols 2 occur. Thus $\text{zeros}(r) = \{2, 6\}$ for the second 012-row r in (1). Intervals in the Boolean lattice $\{0, 1\}^w$ and 012-rows² are the same thing. For instance the interval of all eight bitstrings u with $(0, 1, 0, 0, 0) \leq u \leq (1, 1, 0, 1, 1)$ equals the 012-row $(2, 1, 0, 2, 2)$. Bitstrings can either be viewed as singleton intervals or as 012-rows r with $\text{twos}(r) = \emptyset$.

At first we treat the ALLSAT problem in an abstract setting (Theorem 1) which e.g. comprises the usual variable-wise branching. Five Corollaries (some new, some recast material) are formulated in this framework. The main thrust of the article however occurs in the second half (Sections 6 to 9). It is dedicated to *clause-wise* branching which often brings about better compression than variable-wise branching.

1.2 Section 2 reviews binary decision diagrams for later purpose. Section 3 discusses two natural ways a 012-row r can relate towards a fixed subset $\text{SpMod}(\varphi)$ of $\text{Mod}(\varphi)$: We call r *feasible* if $r \cap \text{SpMod}(\varphi) = \emptyset$, and *final* if $r \subseteq \text{SpMod}(\varphi)$. We closer investigate the case when φ is given as a CNF. Frequently we will have $\text{SpMod}(\varphi) = \text{Mod}(\varphi)$. Section 4 introduces the core concept of a “row-splitting mechanism” with respect to a given φ and a well-defined subset $\text{SpMod}(\varphi) \subseteq \text{Mod}(\varphi)$. At first this concept may look far-fetched but Theorem 1 shows that $\text{SpMod}(\varphi)$ can be enumerated in polynomial total time whenever φ happens to have a row-splitting mechanism. Section 5 starts by verifying that traditional variable-wise branching fits the hat of Theorem 1. The five ensuing Corollaries could have been proven in traditional jargon, but we embraced the novel framework of Theorem 1. Apart from Corollary 2 (see above) we like to single out Corollary 4 which states the polynomial total time enumerability of $\text{SpMod}(\varphi) = \text{Mod}(\varphi, k)$ (the k -element models) when φ is given by a DNF.

Section 6 initiates the more ‘ground-breaking’ second half of our article by recalling a well-known propositional tautology. Its 2-dimensional visualization looks like the *Flag of Papua* and accompanies us throughout the remainder of the article. The Flag of Papua underlies the clause-wise branching introduced in 6.2. We call our method (which again fits Theorem 1) the

²In previous publications the clumsier name, “ $\{0, 1, 2\}$ -valued rows” was used. While “intervals” (in the lattice-theoretic sense) or “subcubes” or “terms” occur frequently in the Boolean logic literature, our equivalent concept of a 012-row goes the extra (half) mile to make things more visual. That will particularly benefit us in Sections 5 and 6.

clause-wise ALLSAT 012-algorithm. Fed with a CNF φ it returns $\text{Mod}(\varphi)$ as a disjoint union of 012-rows.

In Section 7 we add the *e*-wildcard which is surprisingly powerful notwithstanding its innocent definition: $e \cdots e$ means “at least one 1 here”. Correspondingly the *(clause-wise) ALLSAT e-algorithm*, when fed with a CNF φ , returns $\text{Mod}(\varphi)$ as a disjoint union of 012*e*-rows. While 012-rows match familiar exclusive sums of products (ESOP), their enhancement to 012*e*-rows can be viewed as an “exclusive sum of fancy terms” (ESOFT). Some immediate relations among ESOP, ESOFT, DNNF (see [D]) and BDD are discussed in 7.1, 7.2. More technical details concerning ESOFT follow in 7.3 to 7.5.

Section 8 features numerical experiments carried out with Mathematica implementations of our ALLSAT algorithms (be it variable-wise or clause-wise, be it 012-level or 012*e*-level). We also add BDD’s to the picture in the form of Mathematica’s hardwired command `SatisfiabilityCount`, and consider weighted Boolean functions.

Section 9, with the title “History and envisaged future”, mainly focuses on the ALLSAT *e*-algorithm. As to history, we e.g. acknowledge the rôle of Redelinghuys and Geldenhuys [RG]. As to the future of the ALLSAT *e*-algorithm, its main “competitor” seems to be binary decision diagrams (BDDs). Specifically we compare ESOFT and BDD with respect to these criteria: Their ability to compress $\text{Mod}(\varphi)$ (and also $\text{Mod}(\varphi, k)$), and their ability to settle the equivalence of two Boolean functions. In Subsection 9.3 we briefly review what *specific* CNF’s have already been tackled by the author with wildcards (e.g. Horn CNF’s), and try to forecast what the future has in store.

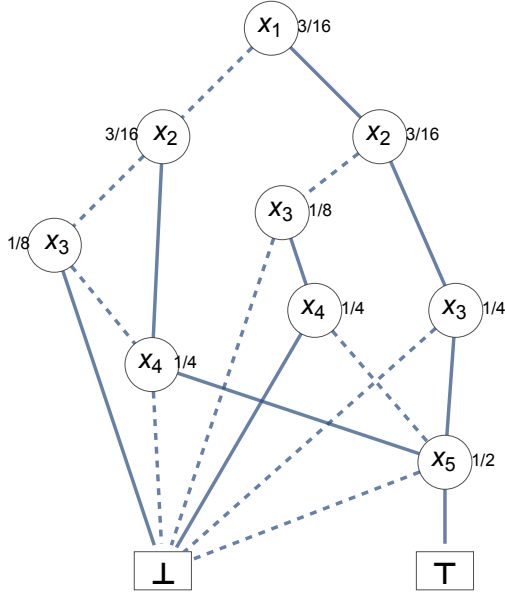
2 A brief revision of BDD’s

We assume a basic familiarity with binary decision diagrams (BDD’s), as e.g. provided by [K]. Section 2 mainly prepares the reoccurrence of BDD’s in Sections 7 to 9.

2.1 Consider the Boolean function $\varphi_1 : \{0, 1\}^5 \rightarrow \{0, 1\}$ that is defined by the BDD in Figure 1. Whether a bitstring u belongs to $\text{Mod}(\varphi_1)$ can be decided as follows. The dashed and solid lines descending from a node labelled x_i are chosen according to whether the i -th component u_i of u is 0 or 1. Thus, in order to decide whether $u = (u_1, u_2, u_3, u_4, u_5) = (0, 1, 0, 1, 0)$ belongs to $\text{Mod}(\varphi_1)$ we follow the dashed line from the root (= top node) x_1 to the node x_2 (since $u_1 = 0$). Then from x_2 with the solid line to x_4 (since $u_2 = 1$), then from x_4 with the solid line to x_5 (since $u_4 = 1$), then from x_5 with the dashed line to \perp (since $u_5 = 0$). The value of u_3 is irrelevant in all of this.

2.2 There is a standard bottom-up way to calculate for each nonleaf node u the probability p_u that a random bitstring fed to u (viewed as the root of an *induced* BDD) triggers a path to \top . Namely, p_u is simply the arithmetic mean of the probabilities attached to the sons of u . This is carried out in Figure 1. In particular $p_u = \frac{3}{16}$ for the root $u = x_1$ implies that $\frac{3}{16} \cdot 2^5 = 6$ bitstrings from $\{0, 1\}^5$ are models of φ_1 .

Figure 1: Some BDD



2.3 In a similar vein the models themselves can be enumerated. Namely, for each branching node u of a BDD of ψ let ψ_u be the Boolean function defined by the induced BDD with root u . The enumerations of all set $\text{Mod}(\psi_u)$ are obtained recursively in straightforward ways. For instance, if $\psi = \varphi_1$ then (recursively or ad hoc) one sees that the enumerations of the sets $\text{Mod}(\psi_u)$ belonging to the three nodes u labelled by x_3 are, from left to right:

$$(2) \quad (0, 1, 1), (1, 0, 1), (1, 2, 1)$$

From (2) (and from (1, 1) matching the leftmost x_4) one gets the sets $\text{Mod}(\psi_u)$ belonging to the two nodes labelled by x_2 :

$$(3) \quad (\mathbf{0}, 0, 1, 1) \uplus (\mathbf{1}, \mathbf{2}, 1, 1), \quad (\mathbf{0}, 1, 0, 1) \uplus (\mathbf{1}, 1, 2, 1)$$

From (3) one finally obtains

$$(4) \quad \text{Mod}(\varphi_1) = (\mathbf{0}, 0, 0, 1, 1) \uplus (\mathbf{0}, 1, 2, 1, 1) \uplus (\mathbf{1}, 0, 1, 0, 1) \uplus (\mathbf{1}, 1, 1, 2, 1)$$

Generally the degree of compression in the enumeration of $\text{Mod}(\varphi)$, i.e. the number of don't-cares 2 in the 012-rows, depends on the *index gaps* $j - i$ of directed edges $x_i \rightarrow x_j$. In the worst case, when all index gaps are 1, there is no compression, i.e. all 012-rows are 01-rows = bitstrings.

3 Feasibility and finality of 012-rows

Let $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ be any Boolean function and let $SpMod \subseteq Mod(\varphi)$ be any class of “special” models. Call any 012-row r of length w *feasible* if $r \cap SpMod \neq \emptyset$. In Section 3 we only consider the case $SpMod = Mod(\varphi)$. Then the feasibility of r just means that the partial variable assignment defined by $zeros(r)$ and $ones(r)$ extends to a model of φ . It will sometimes be useful to look at this condition from a slightly different angle. Namely, as indicated in footnote 1 each 012-row r matches a unique term $T(r)$ as follows: If say $r = (2, 1, 2, 0, 0)$ then $T(r) = x_2 \wedge \bar{x}_4 \wedge \bar{x}_5$. Conversely each term T matches a 012-row $r(T)$. The above condition $r \cap Mod(\varphi) \neq \emptyset$ thus amounts to say that the Boolean function $T(r) \wedge \varphi$ is satisfiable. A (*row*) *feasibility test* is a subroutine which, when fed with a 012-row r , produces an answer *yes* or *no*. The feasibility test is *weak* if (“no” $\Rightarrow r$ is infeasible), and it is *perfect*³ if additionally (“yes” $\Rightarrow r$ is feasible). A *finality test* (with respect to a class of Boolean functions φ) is a subroutine which, when fed with φ and a 012-row r , produces an answer *yes* or *no*. The finality test is *weak* if (“no” $\Rightarrow r$ is not final), and it is *perfect* if additionally (“yes” $\Rightarrow r$ is final).

Feasibility and finality of 012-rows r will be crucial in our approach to enumerate $Mod(\varphi)$. Deciding whether r is φ -feasible or φ -final depends a lot on how φ is presented, and can be difficult. In 3.1 to 3.3 we illustrate the two concepts when φ is rendered as a CNF.

3.1 Let us illustrate feasibility in the case where $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ is given as a CNF with clauses C_1, \dots, C_h . Here it is more succinct to label the positions of a 012-row r by x_1, \dots, x_w instead of just $1, 2, \dots, w$. Thus if $w = 7$ and $r = (2, 0, 0, 1, 1, 2, 0)$ it holds that $\{x_2, x_3\} \subseteq zeros(r)$. As to the clauses, let us identify C_i with the set of literals appearing in it. Furthermore we write C_i^+ for the set of positive literals occurring in C_i , and C_i^- for the set of *negated* negative literals occurring in C_i . Thus if C_i “was” $x_3 \vee x_5 \vee \bar{x}_6 \vee \bar{x}_9$ then now $C_i = \{x_3, x_5, \bar{x}_6, \bar{x}_9\}$ and $C_i^+ = \{x_3, x_5\}$ and $C_i^- = \{\bar{x}_6, \bar{x}_9\}$ (not $\{\bar{x}_6, \bar{x}_9\}$). Suppose C_i and r are such that

$$(5) \quad C_i^+ \subseteq zeros(r) \text{ and } C_i^- \subseteq ones(r).$$

Then obviously r is infeasible (with respect to φ). But r can be infeasible without there being a clause C_i ($1 \leq i \leq h$) satisfying (5). We thus get a weak feasibility test, which we call Test 1. (It appears to be “very weak” but if always $C_i^- = \emptyset$ as in 6.4, it becomes perfect.)

Consider now clauses C_i and C_j ($i \neq j$) such that $x_p \in C_i^+ \cap C_j^-$ and

$$(6) \quad (C_i^+ \setminus \{x_p\}) \cup C_j^+ \subseteq zeros(r) \text{ and } (C_j^- \setminus \{\bar{x}_p\}) \cup C_i^- \subseteq ones(r).$$

Then each bitstring $u \in r$ that wants to satisfy C_i must have 1 at position x_p because by (6) no other literal in C_i can satisfy C_i . Likewise, if u wants to satisfy C_j then it needs to have 0 at position x_p . It follows again that row r is infeasible. The ensuing weak feasibility test we call Test 2. Other weak feasibility tests along the same lines can be designed but for us Test 1 and (to lesser extent) Test 2 will suffice.

3.2 Fix $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$. Then a 012-row r of length w is *final* (w.r.t. φ) if $r \subseteq Mod(\varphi)$. Say $\varphi = C = \{x_3, x_5, \bar{x}_6, \bar{x}_9\}$. If r is a 012-row of length $w \geq 9$ that either satisfies $ones(r) \cap$

³For simplicity we disregard a third option “don’t know”. Consequently a “yes” of a weak feasibility test must not be trusted. Of course, the better the test, the more likely “yes” is correct.

$\{x_3, x_5\} \neq \emptyset$ or $\text{zeros}(r) \cap \{x_6, x_9\} \neq \emptyset$ then $r \subseteq \text{Mod}(\varphi)$, and so r is final. Generally if φ is a CNF with clause set $\{C_1, \dots, C_h\}$ such that all clauses C_i behave as C above then $r \subseteq \text{Mod}(C_1) \cap \dots \cap \text{Mod}(C_h) = \text{Mod}(\varphi)$.

Conversely suppose some clause, say $C_j = \{x_3, x_5, \bar{x}_6, \bar{x}_9\}$, misbehaves in that $\text{ones}(r) \cap \{x_3, x_5\} = \emptyset = \text{zeros}(r) \cap \{x_6, x_9\}$ for some row r . To fix ideas assume $x_3 \in \text{zeros}(r), x_5 \in \text{twos}(r), x_6 \in \text{ones}(r), x_9 \in \text{twos}(r)$. Then there is $u \in r$ with $u_5 = 0$ and $u_9 = 1$ which thus violates C_j (since $u_3 \vee u_5 \vee u_6 \vee u_9 = 0 \vee 0 \vee 0 \vee 0 = 0$). Hence $r \not\subseteq \text{Mod}(\varphi)$. To summarize, $r \subseteq \text{Mod}(\varphi)$ if and only if the clauses of φ behave well in the sense defined above. This yields a perfect finality test of cost $O(hw)$ per row.

3.3 As to probability, let $\varphi = \varphi(x_1, \dots, x_w)$ be a random CNF with totally h clauses, each of length λ . Given w and γ , for a random 012-row r of length w with $|\text{twos}(r)| = \gamma$ we denote by $\text{prob}(w, \gamma, h, \lambda)$ the probability that r is final with respect to φ . In order to show

$$(6) \quad \text{prob}(w, \gamma, h, \lambda) = [1 - (0.5)^{\frac{\lambda}{w}(w-\gamma)}]^h$$

fix an arbitrary clause C of φ . Then the probability that a random $i \in [w]$ both belongs to $[w] \setminus \text{twos}(r)$, and is an index of a literal of C , is $q = \frac{w-\gamma}{w} \cdot \frac{\lambda}{w}$. (Clearly the two events are independent.) It follows that the expected overlap of C with $\text{ones}(r) \cup \text{zeros}(r)$ has length qw . So the probability that at least one bit satisfies C is $p = 1 - (0.5)^{qw}$. Hence p^h is the probability that all clauses behave well, which (as seen before) is equivalent to $r \subseteq \text{Mod}(\varphi)$. See also 8.1.

4 The row-splitting mechanism per se

Throughout the article we tackle ALLSAT (independent of how φ is given) by constructing a search tree \mathcal{T} rooted at $\{0, 1\}^w = (2, 2, \dots, 2)$ in a (preorder) depth-first manner, such that all nodes are 012-rows and such that the union of leaves equals $\text{Mod}(\varphi)$. Our search tree \mathcal{T} needs not be binary, thus a branching node (= non-leaf) may have more than two nodes.

4.1 Specifically, \mathcal{T} arises whenever a Boolean function $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ enjoys a *row-splitting mechanism* with respect to a fixed subset $Sp\text{Mod} \subseteq \text{Mod}(\varphi)$ and with respect to positive integers $h = h(\varphi), d = d(\varphi), s = s(\varphi)$ (the latter $\geq w$ for convenience). That means three things. First, coupled to each 012-row r (always of length w) is an integer $\deg(r) \in [0, h]$, called the *degree* of r . We postulate that $(2, 2, \dots, 2)$ has degree 0 and that generally $\deg(r)$ can be calculated in $\leq d$ time. Second, the definition of degree is such that each feasible 012-row of degree h is final. Third, each feasible 012-row r of degree $< h$ can *split* (whenever desired) in $\leq s$ time into τ many disjoint 012-rows (its *sons*). Here $\tau \geq 1$ and the sons are ordered as first (r_1) , second (r_2) , \dots , last (r_τ) . Furthermore it holds that

(7a) all sons r_i are feasible;

(7b) all sons r_i have $\deg(r_i) > \deg(r)$;

(7c) $(r_1 \uplus \dots \uplus r_\tau) \cap Sp\text{Mod} = r \cap Sp\text{Mod}$.

The proof of Theorem 1 will show that whenever a row-splitting mechanism with respect to SpMod exists then SpMod is necessarily nonempty.

Theorem 1: There is an algorithm which for each Boolean function φ with a row-splitting mechanism with respect to SpMod enumerates SpMod , using R many disjoint 012-rows, in polynomial total time $O(Rh(d + s))$.

Proof: We construct a sequence of growing trees \mathcal{T}_i whose nodes are 012-rows. The leaves of \mathcal{T}_i will be referred to as *temporary* leaves, and their degrees are recorded. The first tree \mathcal{T}_1 has a single node $(2, 2, \dots, 2)$ which is simultaneously its root and only temporary leaf. The last \mathcal{T}_i is our \mathcal{T} from above. The step from \mathcal{T}_j to \mathcal{T}_{j+1} is common⁴ preorder depth-first search, the preorder being determined by the postulated ordering of sons of splitting rows

Specifically, due to (7a) the splitting mechanism can be continued on *each* temporary leaf of degree $< h$. It further follows from (7c) and induction that the union of all temporary leaves is always disjoint and *contains* $\text{Mod}(\varphi)$. From (7b) follows that eventually all temporary leaves have degree h , i.e. are leaves of \mathcal{T} . Because all leaves (being final by assumption) are contained in SpMod , the union of all leaves *equals* SpMod . In particular, since 012-rows are never empty, it follows that $\text{SpMod} \neq \emptyset$.

As to the cost analysis, since there are $|\mathcal{T}| - R$ many nonleaves, and they are bijective to the occurred row-splittings, the cost of the latter amounts to $O((|\mathcal{T}| - R)s) = O(|\mathcal{T}|s)$. There is no other hidden cost such as pruning infeasible rows. Because the depth of \mathcal{T} is $\leq h$ we get $|\mathcal{T}| \leq Rh$. By the above the total cost of calculating degrees is $O(Rhd)$. Furthermore, stacking or outputting a (final) length w bitstring costs $O(w)$. Hence, in view of $w \leq s$, the overall cost is $O(Rhd) + O(Rhs) + O(Rhw) = O(Rh(d + s))$. \square

Unfortunately, all that can be *proven* about the number R of 012-rows is that $0 < R \leq |\text{SpMod}|$. Here $<$ holds because the proof above implied that $\text{SpMod} \neq \emptyset$ whenever φ has a row-splitting mechanism with respect to SpMod . And \leq is due to the disjointness of rows. Practise shows (Section 8) that $R \ll |\text{SpMod}|$ occurs frequently. In our applications of Theorem 1 it will always be that $r_1 \uplus \dots \uplus r_t$ in (7c) is actually a *subset* of r . Yet this is not required in the proof of Theorem 1.

5 Variable-wise branching in new guises

One natural approach to determine $\text{Mod}(\varphi)$ for a Boolean function $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ is to fix some variable-ordering (for us that will always be x_1, x_2, \dots) and to combine variable-wise branching with a SAT-solver in a recursive manner. In 5.1 we carefully check that this basic method⁵ induces a particular kind of row-splitting mechanism, one which necessarily delivers $\text{Mod}(\varphi)$ one-by-one. Subsections 5.2 to 5.4 feature various enhancements of the basic method.

⁴From a psychology point of view I prefer to *construct* \mathcal{T} rather than search within an already existing tree \mathcal{T} . In fact, the equivalent (but a bit antiquated) framework of a last-in-first-out stack is more appealing to me than depth-first search of a tree. It will be adapted in Section 6.

Namely, in 5.2 we show how $\text{Mod}(\varphi^c)$ can be obtained from $\text{Mod}(\varphi)$ in compressed fashion and in polynomial total time. Here φ^c is defined by $\text{Mod}(\varphi) \uplus \text{Mod}(\varphi^c) = \{0, 1\}^w$. In 5.3 we switch from individual φ 's to certain “hereditary” classes \mathcal{C} of Boolean functions. In 5.4 we additionally replace $\text{SpMod} = \text{Mod}(\varphi)$ by $\text{SpMod} = \text{Mod}(\varphi, k)$. Two instances fitting this framework are treated in Corollary 4 (φ 's in DNF format) and Corollary 5 (φ 's encoding hypergraphs).

5.1 Let us see how usual variable-wise branching for Boolean functions $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ (in arbitrary format) fits the bill of Theorem 1. Here $h = w$ and the row-splitting mechanism works as follows. By definition the degree of a 012-row r is $\deg(r) := \min(\text{twos}(r)) - 1$. For instance $r = (0, 1, 1, 0, 1, 2, 1, 0, 2, 1)$ has $\deg(r) = 5$, and $\deg((2, 0, 2)) = 0$. Thus $d(\varphi) = w$ and feasible 012-rows of degree w (i.e. bitstrings) are indeed final. Suppose r is feasible and $q := \deg(r) < w$. Let ρ_0 and ρ_1 be the 012-rows arising from r by substituting the 2 at position $q + 1$ by 0 and 1 respectively. The fact that $\deg(\rho_0) = \deg(\rho_1) > \deg(r)$ is akin to (7b), and $r = \rho_0 \uplus \rho_1$ is akin to (7c). However, either ρ_0 or ρ_1 (but not both since r is feasible) may be infeasible, which would clash with (7a). We thus need⁶ a satisfiability subroutine which decides matters. If say only ρ_0 is feasible then (7a) holds with $\tau = 1$ and $r_1 := \rho_0$. Obviously $r_1 \cap \text{SpMod}(\varphi) = r \cap \text{SpMod}(\varphi)$, and so (7b) and (7c) hold as well.

If the branching is variable-wise and the definition of $\deg(r)$ as above then clearly only 012-rows of type $(*, \dots, *, 2, \dots, 2)$ with $* \in \{0, 1\}$ will ever be subject to row-splitting. We dub this method the *variable-wise ALLSAT 012-algorithm*. We will see it in action in the remainder of Section 5.

5.2 Each set system $\mathcal{F} \subseteq \mathcal{P}[w]$ invites two kinds of complementation. The *global complement* $\mathcal{F}^c := \mathcal{P}[w] \setminus \mathcal{F}$, and the *member-wise complement* $\mathcal{F}_c := \{[w] \setminus X : X \in \mathcal{F}\}$. Each \mathcal{F} equals $\text{Mod}(\varphi)$ for a unique Boolean function $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$. Hence $\mathcal{F}^c = \text{Mod}(\varphi^c)$ for $\varphi^c(x_1, \dots, x_n) := \varphi(x_1, \dots, x_n)$; and $\mathcal{F}_c = \text{Mod}(\varphi_c)$ for $\varphi_c := \varphi(\bar{x}_1, \dots, \bar{x}_n)$. For instance, if $\varphi = \varphi(x_1, x_2, x_3) = x_1 \wedge (\bar{x}_2 \vee x_3)$ then $\varphi^c = x_1 \wedge (\bar{x}_2 \vee x_3) = \bar{x}_1 \vee (x_2 \wedge \bar{x}_3)$ and $\varphi_c = \bar{x}_1 \wedge (x_2 \vee \bar{x}_3)$. Evidently $(\varphi^c)^c = (\varphi_c)_c = \varphi$.

Consider the enumeration $\text{Mod}(\varphi) = r_1 \uplus \dots \uplus r_t$. Defining \bar{r}_i by $\text{ones}(\bar{r}_i) := \text{zeros}(r_i)$, $\text{zeros}(\bar{r}_i) := \text{ones}(r_i)$, $\text{twos}(\bar{r}_i) := \text{twos}(r_i)$, evidently yields the enumeration $\text{Mod}(\varphi_c) = \bar{r}_1 \uplus \dots \uplus \bar{r}_t$. What about $\text{Mod}(\varphi^c)$? Getting the cardinality is easy:

$$|\text{Mod}(\varphi^c)| = 2^w - |\text{Mod}(\varphi)| = 2^w - |r_1| - |r_2| - \dots - |r_t|,$$

yet finding an enumeration of $\text{Mod}(\varphi^c)$ (even just one-by-one) is harder. Of course the naive approach to pick all $u \in \{0, 1\}^w$ and check whether or not $u \in \text{Mod}(\varphi)$, does not yield a polynomial total time enumeration of $\text{Mod}(\varphi^c)$. The issue will be tackled in Corollary 1 (with the rôles of φ and φ^c switched). Thus suppose an enumeration of $\text{Mod}(\varphi^c)$ happens to be known. How this “happens” needs not concern us here⁷

⁶Whether the row-splitting mechanism is induced by variable-wise branching or something else, in order to fulfil (7a) one *always* needs a SAT-solver. More precisely, (7a) is essential for any *theoretic* cost analysis. Fortunately, these days SAT-solvers are very efficient [Knuth 2015], at least for $\text{SpMod} = \text{Mod}(\varphi)$.

⁷If e.g. φ is given as a CNF then a DNF of φ^c is readily obtained by applying de Morgan's laws. For special φ^s this DNF is orthogonal already. In any case, there is a large research body of how to make DNF's orthogonal. See also the remarks after Corollary 3.

Corollary 1: Suppose that for the Boolean function $\varphi : \{0,1\}^w \rightarrow \{0,1\}$ an enumeration of $\text{Mod}(\varphi^c)$ is known which uses t many disjoint 012-rows. Then $\text{Mod}(\varphi)$ can be enumerated, using R many disjoint 012-rows, in time $O(Rtwh)$.

Proof: In view of Theorem 1 it suffices to exhibit a row-splitting mechanism for φ with (per row) *splitting time* $s(\varphi) = O(wt)$ because then $O(Rh(d+s)) = O(Rh(w+wt)) = O(Rhwt)$. Suppose that our given enumeration is $\text{Mod}(\varphi^c) = r'_1 \uplus \dots \uplus r'_t$. Recall that in 5.1 we considered a feasible r and a decomposition $r = \rho_0 \uplus \rho_1$ induced by variable-wise branching. The feasibility of ρ_0 (similarly ρ_1) is equivalent to $\rho_0 \cap \text{Mod}(\varphi) \neq \emptyset$, which amounts to $\rho_0 \not\subseteq \text{Mod}(\varphi^c)$, which amounts to $|\rho_0 \cap \text{Mod}(\varphi^c)| < |\rho_0|$. This inequality can be tested because we have $|\rho_0| = 2^\gamma$ where $\gamma := \lfloor \text{twos}(\rho_0) \rfloor$, and

$$\rho_0 \cap \text{Mod}(\varphi^c) = (\rho_0 \cap r'_1) \uplus (\rho_0 \cap r'_2) \uplus \dots \uplus (\rho_0 \cap r'_t).$$

If $\text{zeros}(\rho_0) \cap \text{ones}(r'_i) \neq \emptyset$ or $\text{ones}(\rho_0) \cap \text{zeros}(r'_i) \neq \emptyset$ then $\rho_0 \cap r'_i = \emptyset$. Otherwise $\rho_0 \cap r'_i$ can again be written as a 012-row. For instance $(0, 1, 2, 2, 1, 2) \cap (2, 1, 2, 0, 2, 2) = (0, 1, 2, 0, 1, 2)$. It follows that $|\rho_0 \cap \text{Mod}(\varphi^c)|$ can be calculated in $O(wt)$ time. A notable special case of ρ_0 being feasible, i.e. satisfying $|\rho_0 \cap \text{Mod}(\varphi^c)| < |\rho_0|$, is that $|\rho_0 \cap \text{Mod}(\varphi^c)| = 0$. This amounts to $\rho_0 \subseteq \text{Mod}(\varphi)$, i.e. to the finality of ρ_0 . The enumeration of $\text{Mod}(\varphi)$ can thus entail proper 012-rows ρ_0 . \square

5.3 Here we switch from individual φ 's to classes \mathcal{C} of Boolean functions. Albeit things possibly generalize, we demand that each $\psi \in \mathcal{C}$ is given by a *Boolean formula* and not by some other gadget. Further we view $\psi(x_1, x_2, x_4) := (x_1 \rightarrow x_2) \wedge x_4$ as distinct from $\psi_0(x_1, x_2, x_3, x_4) := (x_1 \rightarrow x_2) \wedge x_4$ because they are of type $\{0,1\}^3 \rightarrow \{0,1\}$ and $\{0,1\}^4 \rightarrow \{0,1\}$ respectively. Accordingly we have *arities* $|\psi| = 3$ and $|\psi_0| = 4$. We call a class \mathcal{C} of Boolean functions *hereditary* if for each $\varphi \in \mathcal{C}$ the substitution⁸ of variables with 0 or 1 yields again a function from \mathcal{C} .

Corollary 2: Let \mathcal{C} be a hereditary class of Boolean functions and suppose the satisfiability of each arity w member of \mathcal{C} can be tested in time $\leq \text{sat}(w)$, where sat is a monotone function. Then for each $\varphi \in \mathcal{C}$ with $|\varphi| = w$ one can use variable-wise branching to enumerate $\text{Mod}(\varphi)$ in $O(R\text{wsat}(w))$ time as a disjoint union of R many 012-rows.

Proof: In view of Theorem 1 and $h = w$ and $d + s = w + s \leq 2s$ it suffices to show that each $\varphi \in \mathcal{C}$ possesses a row-splitting mechanism with $s(\varphi) = O(\text{sat}(w))$. As seen above, splitting rows r amounts to checking the feasibility of rows ρ , and this in turn reduces to the satisfiability of $\varphi \wedge T(\rho)$. Since \mathcal{C} is hereditary, $\psi = \varphi \wedge T(\rho)$ belongs to \mathcal{C} . Because of $|\psi| \leq |\varphi| = w$ its satisfiability is testable in time $\text{sat}(w)$. \square

5.4 In the sequel we content ourselves to consider $\text{SpMod}(\varphi) = \text{Mod}(\varphi, k) := \{u \in \text{Mod}(\varphi) : |u| = k\}$. Here the *cardinality* of a bitstring $u \in \{0,1\}^w$ is the number of 1's, thus $|u| := |\{i \in [w] : u_i = 1\}|$. A Boolean function φ is *k-satisfiable* if $\text{Mod}(\varphi, k) \neq \emptyset$. (Do not confuse $|u|$ and $|\varphi|$.)

⁸For instance, take $\varphi(x_1, x_2, x_3, x_4, x_5) = (x_2 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge x_3$ of arity 5. Then the substitution $\{x_2 \rightarrow 1, x_5 \rightarrow 0\}$ yields the arity 3 function $\psi(x_1, x_3, x_4) = (x_1 \vee x_4) \wedge x_3$. The substitution $\{x_3 \rightarrow 0\}$ yields the arity 0 zero function $\psi_0(x_1, x_2, x_4, x_5) = 0$.

Corollary 3: Let \mathcal{C} be a hereditary class of Boolean functions and suppose the k -satisfiability of each $\psi \in \mathcal{C}$ can be tested in time $\text{sat}(w, k)$ where sat is monotone in each component. Then for each $\varphi \in \mathcal{C}$ with $|\varphi| = w$ and each $k > 0$ one can enumerate $\text{Mod}(\varphi, k)$ in $O(R\text{wsat}(w, k))$ time where $R = |\text{Mod}(\varphi, k)|$.

Proof: A row ρ is k -feasible if $\rho \cap \text{Mod}(\varphi, k) \neq \emptyset$. Fine-tuning the previous proof we need to show that checking the k -feasibility of a 012-row ρ can be done in time $\text{sat}(w, k)$. If $\kappa := |\text{ones}(\rho)|$ is $> k$ then ρ is not k -feasible. Otherwise consider $\psi = \varphi \wedge T(\rho)$. Because $(|\psi|, k - \kappa) \leq (w, k)$ one can check in time $\text{sat}(|\psi|, k - \kappa) \leq \text{sat}(w, k)$ whether ψ is $(k - \kappa)$ -satisfiable. If yes then ρ is k -feasible, otherwise not. \square

In Corollaries 3 to 5 the constant within $O(\dots)$ is independent of k . As opposed to $r \subseteq \text{Mod}(\varphi)$ obviously $r \subseteq \text{Mod}(\varphi, k)$ is only possible when $\text{twos}(r) = \emptyset$. Hence the enumeration in Corollary 3 is necessarily⁹ one-by-one, i.e. $R = |\text{Mod}(\varphi, k)|$. The same remark applies to Corollaries 4, 5 and even 1, 2 which can also be reformulated with $\text{Mod}(\varphi, k)$ instead of $\text{Mod}(\varphi)$.

Corollary 4: If $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ is given as DNF with t terms then $\text{Mod}(\varphi, k)$ can be enumerated in $O(Rtw^2)$ time where $R = |\text{Mod}(\varphi, k)|$.

Proof: In view of Corollary 3 it suffices to show that for DNF's φ the time $\text{sat}(w, k)$ to test for $r \cap \text{Mod}(\varphi, k) = \emptyset$ is $O(tw)$. For starters, if $\{T_1, \dots, T_t\}$ is the set of terms of φ then $\text{Mod}(\varphi) = r(T_1) \cup \dots \cup r(T_t)$. Hence $r \cap \text{Mod}(\varphi, k) \neq \emptyset$ iff some set $r \cap r(T_i)$ contains a bitstring u with $|u| = k$. Now $r \cap r(T_i) = \emptyset$ iff $\text{ones}(r) \cap \text{zeros}(r(T_i)) \neq \emptyset$ or $\text{zeros}(r) \cap \text{ones}(r(T_i)) \neq \emptyset$. If $r \cap r(T_i) \neq \emptyset$ then $\rho_i := r \cap r(T_i)$ can be written as 012-row (as seen in the proof of Corollary 1). Evidently ρ_i contains at least one u with $|u| = k$ iff $|\text{ones}(\rho_i)| \leq k \leq |\text{ones}(\rho_i)| + |\text{twos}(\rho_i)|$. It follows that $\text{sat}(w, k) = O(tw)$. \square

As an application of Corollary 4 it follows at once that the k -faces of a simplicial complex given by its facets can be enumerated in polynomial total time [W11].

Recall that a k -hitting set of a hypergraph $\mathcal{H} \subseteq \mathcal{P}[w]$ is a k -element set X such that $X \cap Y \neq \emptyset$ for all hyperedges $Y \in \mathcal{H}$. A hypergraph is of rank d if $|Y| \leq d$ for all $Y \in \mathcal{H}$. Here $\mathcal{P}[w]$ is the powerset of $[w]$.

Corollary 5: All R many k -hitting sets of a rank 3 hypergraph $\mathcal{H} \subseteq \mathcal{P}[w]$ can be enumerated in time $O(Rw(1.6316^k + kw))$.

Proof: Coupled to $\mathcal{H} \subseteq \mathcal{P}[w]$ consider the Boolean function $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ in CNF whose h clauses match the edges of \mathcal{H} . Thus the edge $Y = \{1, 3, 4\} \in \mathcal{H}$ matches the clause $x_1 \vee x_3 \vee x_4$. The class \mathcal{C} of all such (positive) CNF's is hereditary, as is illustrated in footnote 6. According to [Wa] one can test in time $\text{sat}(w, k) = O(1.6316^k + kw)$ whether a rank 3 hypergraph $\mathcal{H} \subseteq \mathcal{P}[w]$

⁹The verdict “necessarily” is a bit too harsh: There is a way to compress $\text{Mod}(\varphi, k)$ using suitable wildcards. We don't dwell on that in the present article but see 9.3.3 for some hints.

has a k -hitting set. Since *sat* is monotone the claim follows from Corollary 3. \square

It is interesting to compare Corollary 5 with this fixed-parameter result which doesn't use feasibility checks in its proof and which follows at once from Lemma 1.7 in [FG]: All R many k -hitting sets of a rank 3 hypergraph \mathcal{H} can be enumerated in time $O(3^k \cdot k \cdot \|\mathcal{H}\|)$ where $\|\mathcal{H}\| := w + \sum\{|X| : X \in \mathcal{H}\}$. As to the benefit of Corollary 5, notice that possibly $R \ll 3^k$.

5.5 Let again $\mathcal{H} \subseteq \mathcal{P}[w]$ be a hypergraph and $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ such that $\mathcal{H} = \text{Mod}(\varphi)$. If now $\text{SpMod}(\varphi)$ is the family of inclusion-minimal hitting sets of \mathcal{H} , can one check the feasibility of a 012-row in time polynomial in $\|\mathcal{H}\|$? If yes, Corollary 3 would yield a polynomial total time algorithm for enumerating $\text{SpMod}(\varphi)$, thus settling a long-standing open problem.

6 CNF-ALLSAT using clause-wise branching

There is an alternative to the variable-wise branching we applied so far. But it applies only to CNF-ALLSAT, i.e. the Boolean function $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ comes as CNF with h clauses C_i . Then we can design a row-splitting mechanism (Section 4) in such a way that each 012-row r has its own particular “pending” clause C_i that needs to be “imposed”. The potentially more than two sons ρ_j of r are constructed in a way that they all satisfy C_i . This novel kind of clause-wise branching has been programmed by the author in various¹⁰ settings. One benefit of clause-wise branching is that it often delivers fat 012-leaves of the search tree \mathcal{T} (Sec. 4) due to a *gratuitous* finality test: A node of \mathcal{T} is final (i.e. a leaf) if and only if all constraints C_i have been imposed on it. The detailed example in 6.2 prepares the ground for Theorem 2.

6.1 We split the model set of any single clause $y_1 \vee y_2 \vee \dots \vee y_m$ as follows into a disjoint union of m many 012-rows (here $m = 5$):

y_1	y_2	y_3	y_4	y_5
1	2	2	2	2
0	1	2	2	2
0	0	1	2	2
0	0	0	1	2
0	0	0	0	1

Table 1: *The Flag of Papua*

As introduced in [W4, Sec.3], we shall refer to the pattern in Table 1 as the *Flag of Papua* with its three colors upon, below, and above the diagonal. It matches this familiar tautology (using concatenation instead of \wedge):

$$(8) \quad y_1 \vee y_2 \vee y_3 \vee y_4 \vee y_5 \quad \leftrightarrow \quad y_1 \vee \bar{y}_1 y_2 \vee \bar{y}_1 \bar{y}_2 y_3 \vee \bar{y}_1 \bar{y}_2 \bar{y}_3 y_4 \vee \bar{y}_1 \bar{y}_2 \bar{y}_3 \bar{y}_4 y_5$$

6.2 To develop the details of clause-wise branching, consider the CNF

¹⁰In previous publications I spoke of a “principle of exclusion” instead of more telling “clause-wise branching”. A crisp Master Theorem (as Theorem 2 below) was lacking.

$$(9) \quad \varphi_2 := (\bar{x}_1 \vee \bar{x}_2 \vee x_3)(x_2 \vee \bar{x}_3 \vee \bar{x}_4)(\bar{x}_3 \vee x_5)(x_1 \vee x_3 \vee x_5)(x_1 \vee x_4 \vee \bar{x}_5)(\bar{x}_1 \vee x_2 \vee x_3)(\bar{x}_2 \vee x_4 \vee x_5).$$

It will pay to introduce some redundancy in that our 012-rows r are indexed by $x_1, \bar{x}_1, \dots, x_5, \bar{x}_5$ rather than just x_1, \dots, x_5 as in 3.1. Hence it suffices to speak of C_i (omitting C_i^+, C_i^-), but it entails that when in r the x_i -component is 1, its \bar{x}_i -component is 0, and vice versa. An entry 2 at the x_i -component forces the \bar{x}_i -component to be 2 as well. Thus such entries are free to be 0 or 1, but in coordination.

Let us denote by C_j the j -th clause in (9). We identify C_j with the set of its literals, thus $C_2 = \{x_2, \bar{x}_3, \bar{x}_4\}$. Let Mod_i be the set of all length 10 bitstrings that satisfy C_1, C_2, \dots, C_i , i.e. the simultaneous *models* of these clauses. Following the “principle of exclusion” (footnote 8), starting with $\text{Mod}_0 = \{0, 1\}^{10}$ we shall inductively sieve Mod_{i+1} from Mod_i (thus *exclude* the duds from Mod_i) until we arrive at the model set Mod_7 of $\varphi_2(x_1, \dots, x_5)$. We say a 012-row r *satisfies a clause* C_i if *all* bitstrings $u \in r$ satisfy C_i , and r *violates* C_i if *at least one* $u \in r$ violates C_i . In formulas, $r \subseteq \text{Mod}(C_i)$ respectively $r \not\subseteq \text{Mod}(C_i)$.

In Table 2 below Mod_0 is encoded by row r_1 , and Mod_1 is displayed as the disjoint union of r_2, r_3, r_4 . The columns of the boldface Flag of Papua within r_2 to r_4 match the literals $\bar{x}_1, \bar{x}_2, x_3$ of C_1 . The rows r_2 to r_4 constitute our *working stack*, of which always the top row will be treated¹¹. To do so we keep track of which clause is pending for each row. For instance $PC = 3$ for r_4 since besides C_1 (by construction) r_4 also happens to satisfy C_2 , and so the “pending clause” is C_3 . Next the top row r_2 is split into three Flag of Papua *candidate sons* r_5 to r_7 according to clause C_2 . The new top row is r_5 which in view of $PC = 3$ gives way to the candidate sons r_8 and r_9 . Imposing $C_4 = \{x_1, x_3, x_5\}$ upon r_8 results in $r_{10} = (0, 1, 1, 0, 0, 1, 2, 2, 1, 0)$, which has $PC = 5$. Imposing $C_5 = \{x_1, x_4, \bar{x}_5\}$ upon r_{10} yields $r_{11} = \{0, 1, 1, 0, 0, 1, 1, 0, 1, 0\}$ which happens to be *final*, i.e. r_{11} also satisfies C_6, C_7 . We thus remove r_{11} (which condenses to $(x_1, x_2, x_3, x_4, x_5) = (0, 1, 0, 1, 1)$) from the working stack and put it on a save place (call it the *final stack*). When continuing in this manner until the working stack is empty, the rows in the final stack will provide the sought partitioning of $\text{Mod}(\varphi_2)$. The reader pondering to carry this out in detail be advised that the enhancement in Section 7 is more exciting.

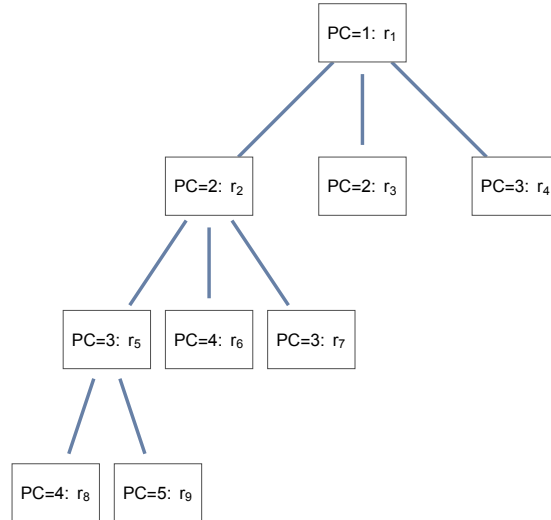
6.2.1 The indicated method begs for a SAT-solver (as in Section 5) in order to immediately get rid of infeasible 012-rows. We dub our procedure the *clause-wise ALLSAT 012-algorithm*, as opposed to the *variable-wise ALLSAT 012-algorithm* from 5.1.

¹¹That method amounts, as in Section 4, to a preorder depth-first search of a tree (Fig.3). As previously mentioned, the equivalent last-in-first-out (LIFO) stack framework (Table 2) seems more appealing.

	x_1	\bar{x}_1	x_2	\bar{x}_2	x_3	\bar{x}_3	x_4	\bar{x}_4	x_5	\bar{x}_5	
$r_1 =$	2	2	2	2	2	2	2	2	2	2	$PC = 1$
$r_2 =$	0	1	2	2	2	2	2	2	2	2	$PC = 2$
$r_3 =$	1	0	0	1	2	2	2	2	2	2	$PC = 2$
$r_4 =$	1	0	1	0	1	0	2	2	2	2	$PC = 3$
$r_5 =$	0	1	1	0	2	2	2	2	2	2	$PC = 3$
$r_6 =$	0	1	0	1	0	1	2	2	2	2	$PC = 4$
$r_7 =$	0	1	0	1	1	0	0	1	2	2	$PC = 3$
$r_3 =$	1	0	0	1	2	2	2	2	2	2	$PC = 2$
$r_4 =$	1	0	1	0	1	0	2	2	2	2	$PC = 3$
$r_8 =$	0	1	1	0	0	1	2	2	2	2	$PC = 4$
$r_9 =$	0	1	1	0	1	0	2	2	1	0	$PC = 5$
$r_6 =$	0	1	0	1	0	1	2	2	2	2	$PC = 4$
$r_7 =$	0	1	0	1	1	0	0	1	2	2	$PC = 3$
$r_3 =$	1	0	0	1	2	2	2	2	2	2	$PC = 2$
$r_4 =$	1	0	1	0	1	0	2	2	2	2	$PC = 3$
...											...

Table 2: Snapshots of the working stack of the clause-wise ALLSAT 012-algorithm (for φ_2).

Figure 2: Search tree matching Table 2



Theorem 2: Let \mathcal{C} be a hereditary class of Boolean CNF's such that the satisfiability of each arity w member of \mathcal{C} can be tested in time $\text{sat}(w)$, where sat is a monotone function. Then for each $\varphi \in \mathcal{C}$ having $|\varphi| = w$ and h clauses one can use clause-wise branching to enumerate $\text{Mod}(\varphi)$ with R many disjoint 012-rows in time $O(Rhw(h + \text{sat}(w)))$.

Proof: Recall that $s(\varphi)$ denotes the splitting time per row. We first show that $s(\varphi) = O(w\text{sat}(w))$, and then that $d(\varphi) = O(hw)$. In view of Theorem 1 the complexity then becomes $O(Rh(d+s)) = O(Rh(hw + w\text{sat}(w)))$ as claimed.

As to $s(\varphi)$, by the example above $s(\varphi) = O(w^2 + w\text{sat}(w)) = O(w\text{sat}(w))$. Indeed, imposing a clause of length $\tau \leq w$ upon a 012-row r of length w (i.e. raising the Flag of Papua) costs $O(w\tau) = O(w^2)$. Each of the τ many candidate sons ρ needs to be tested for feasibility. Testing the feasibility of ρ amounts to testing the satisfiability of $\psi = \varphi \wedge T(\rho)$. Since \mathcal{C} is hereditary, ψ belongs to \mathcal{C} and whence its satisfiability is testable in time $\text{sat}(|\psi|) \leq \text{sat}(w)$. As to $d(\varphi)$, the degree of a 012-row r (i.e. its pending clause) is calculated by scanning the h clauses C_i until $C_i \cap \text{ones}(r) = \emptyset$. Hence $d(\psi) = O(hw)$. \square

We mention that e.g. for the class \mathcal{C} of Horn CNF's the bound in Theorem 2 reduces to $O(Rh^2w^2)$. This is established in [W2] (see also 9.3), though in a framework more clumsy than Theorem 2. While one could also enumerate \mathcal{C} in polynomial total time using variable-wise branching, the high compression achieved by clause-wise branching would virtually disappear.

7 Extension to 012e-rows

We now trim the clause-wise ALLSAT 012-algorithm of Section 6 by moving beyond the don't-care symbol 2. The basic idea is to replace any Flag of Papua (Table 1) by the wildcard, or *e-bubble* (e, e, e, e, e) , which *by definition* means “at least one 1 here”.

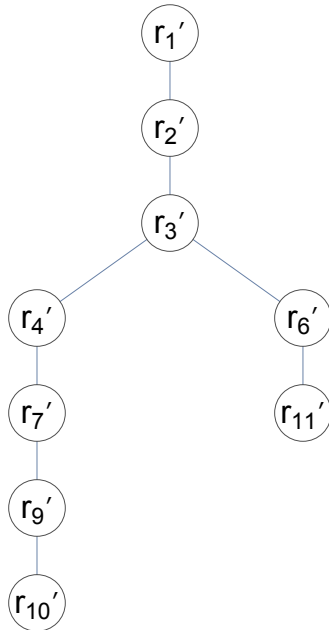
Let us jump into medias res and impose C_1 to C_7 of φ_2 from (9) anew, this time finishing the job. Starting with row $r'_1 = (2, 2, \dots, 2)$ in Table 3. Imposing C_1 upon r'_1 yields r'_2 . Imposing C_2 upon r'_2 yields r'_3 which features a new, disjoint *e-bubble*. The two are distinguished by subscripts. Generally any row featuring the symbols 0, 1, 2 and possibly *e-bubbles* will be called¹² an *012e-row*. In order to impose C_3 upon r'_3 we first partition $r'_3 = r'_4 \uplus r'_5$ as indicated. Notice that the 0 in r'_5 forces the 1 on its left. Furthermore the 0 in r'_5 turns the (e_2, e_2, e_2) in r'_3 to $(e_2, 0, e_2)$, and the 1 in r'_5 turns the (e_1, e_1, e_1) in r'_3 to $(2, 2, 1)$. Similar remarks apply to r'_4 . The advantage of r'_4, r'_5 over r'_3 is that r'_4 satisfies $C_3 = \{\bar{x}_3, x_5\}$ (whence $PC = 4$), and imposing C_3 upon r'_5 immediately yields r'_6 . Notice that r'_6 happens to satisfy C_4 , and so has $PC = 5$. Imposing C_4 upon r'_4 yields r'_7 .

¹²This renames the “ $\{0, 1, 2, e\}$ -valued rows” in previous publications. Of course each 012-row is a 012e-row but not conversely.

	x_1	\bar{x}_1	x_2	\bar{x}_2	x_3	\bar{x}_3	x_4	\bar{x}_4	x_5	\bar{x}_5	
$r'_1 =$	2	2	2	2	2	2	2	2	2	2	$PC = 1$
$r'_2 =$	2	e	2	e	e	2	2	2	2	2	$PC = 2$
$r'_3 =$	2	e_1	e_2	e_1	e_1	e_2	2	e_2	2	2	$PC = 3$
$r'_4 =$	2	e_1	2	e_1	0	1	2	2	2	2	
$r'_5 =$	2	2	e_2	2	1	0	2	e_2	2	2	
$r'_4 =$	2	e_1	2	e_1	0	1	2	2	2	2	$PC = 4$
$r'_6 =$	2	2	e_2	2	1	0	2	e_2	1	0	$PC = 5$
$r'_7 =$	e_2	e_1	2	e_1	0	1	2	2	e_2	2	$PC = 5$
$r'_6 =$	2	2	e_2	2	1	0	2	e_2	1	0	$PC = 5$
$r'_8 =$	1	0	0	1	0	1	2	2	2	2	
$r'_9 =$	0	1	2	2	0	1	2	2	1	0	
$r'_6 =$	2	2	e_2	2	1	0	2	e_2	1	0	$PC = 5$
$r'_{10} =$	0	1	2	2	0	1	1	0	1	0	final
$r'_6 =$	2	2	e_2	2	1	0	2	e_2	1	0	$PC = 5$
$r'_{11} =$	e_1	2	e_2	2	1	0	e_1	e_2	1	0	final

Table 3: The transition from 012-rows to 012e-rows speeds up clause-wise branching

Fig. 3: Search tree for Table 3



Now things get interesting. In order to impose $C_5 = (x_1, x_4, \bar{x}_5)$ upon r'_7 we first partition $r'_7 = r'_8 \uplus r'_9$. Here r'_8 must be deleted since it violates $C_6 = \{\bar{x}_1, x_2, x_3\}$, whereas r'_9 turns to r'_{10} upon imposing C_5 . Actually r'_{10} happens to be final (i.e. satisfies C_6, C_7 as well) and thus is removed from the working stack. Imposing C_5 upon r'_6 yields r'_{11} which again happens to be final. Pinning x_4 in r'_{11} to 1 and 0 respectively shows that

$$(10) \quad r'_{11} = r'_{12} \uplus r'_{13} = (2, 2, 1, 0, 1, 0, \mathbf{1}, 0, 1, 0) \uplus (1, 0, 2, 2, 1, 0, \mathbf{0}, 1, 1, 0).$$

The by construction mutually disjoint rows $r'_{10}, r'_{12}, r'_{13}$, when shrunk back to length 5, yield

$$(11) \quad \text{Mod}(\varphi_2) = (0, 2, 0, 1, 1) \uplus (2, 1, 1, 1, 1) \uplus (1, 2, 1, 0, 1)$$

Comparing (4) and (11) we see that $\text{Mod}(\varphi_1) = \text{Mod}(\varphi_2)$, and so the BDD in Figure 1 yields the same Boolean function as the CNF in (9).

The illustrated method will be dubbed the *ALLSAT e-algorithm*, which sounds better than clause-wise ALLSAT 012e-algorithm. (The adjective “clause-wise” is superfluous since the *e*-formalism does not apply to variable-wise branching). The Subsections 7.1 (alternative terminology) and 7.2 (a first comparison with ESOP, DNNF, BDD) are straightforward whereas Subsections 7.3 to 7.5 are more technical in nature.

7.1 Speaking of 012e-rows is good and well, but occasionally alternative terminology is helpful, e.g. for comparison with some standard formats of Boolean functions. Generalizing ordinary *terms* like $\bar{x}_1 \wedge x_2 \wedge \bar{x}_3$ we hence introduce *fancy terms* like

$$(12) \quad (x_1 \vee x_4) \wedge (x_2 \vee \bar{x}_4) \wedge x_3 \wedge x_5$$

which by definition are (literal-wise) *disjoint* conjunctions of clauses of any length, except that complementary length 1 clauses x_i and \bar{x}_i are forbidden. We shall see in 7.3 that each fancy term is satisfiable. The fancy term in (12) matches r'_{11} in Table 3. Further we call ψ an *exclusive sum of fancy terms* (ESOFT) if $\psi = \psi_1 \vee \dots \vee \psi_m$ is such that

(13a) all ψ_i 's are fancy terms;

(13b) $\text{Mod}(\psi_i) \cap \text{Mod}(\psi_j) = \emptyset$ for all $1 \leq i < j \leq m$.

For instance, the ESOFT corresponding to $r'_{10} \uplus r'_{11}$ in Table 3 is

$$(14) \quad \varphi_3 = \varphi_{31} \vee \varphi_{32} := (\bar{x}_1 \wedge \bar{x}_3 \wedge x_4 \wedge x_5) \vee ((x_1 \vee x_4) \wedge (x_2 \vee \bar{x}_4) \wedge x_3 \wedge x_5).$$

7.1.1 By the correctness of the ALLSAT *e*-algorithm, it holds that $\text{Mod}(\varphi_2) = \text{Mod}(\varphi_3)$. So $\varphi_2 \equiv \varphi_3$ holds in all 2-element Boolean algebras, whence (as is well known) in all Boolean algebras. For instance if $A_1, \dots, A_5 \in \mathcal{B} = \mathcal{P}[S]$ are any subsets of some set S then $\varphi_2 \equiv \varphi_3$ translates, upon putting $\bar{A}_i := S \setminus A_i$, to:

$$(15) \quad (\bar{A}_1 \cup \bar{A}_2 \cup A_3) \cap (A_2 \cup \bar{A}_3 \cup \bar{A}_4) \cap (\bar{A}_3 \cup A_5) \cap (A_1 \cup A_3 \cup A_5)$$

$$\cap (A_1 \cup A_4 \cup \bar{A}_5) \cap (\bar{A}_1 \cup A_2 \cup A_3) \cap (\bar{A}_2 \cup A_4 \cup A_5) =$$

$$(\bar{A}_1 \cap \bar{A}_3 \cap A_4 \cap A_5) \uplus ((A_1 \cup A_4) \cap (A_2 \cup \bar{A}_4) \cap A_3 \cap A_5)$$

7.2 We saw that the ALLSAT e -algorithm turns each CNF into an ESOF. Let us briefly relate ESOF to ESOP, DNNF and BDD. First, ESOF is a powerful generalization of ESOP (see also 7.4.2 and 8.2). Second, ESOF is a special case of Decomposable Negation Normal Form (DNNF). That follows¹³ at once from the definition [D] of DNNF which postulates a disjointness property akin to the definition of “fancy term” in (13a). In contrast ESOF links to BDDs rather by the *other* disjointness property (13b). Indeed, as seen in 2.3, BDDs induce a partition of $\text{Mod}(\varphi)$ into a disjoint union of 012-rows, which hence is a special type of ESOF. A more thorough discussion of ESOF versus BDD follows in Section 9.2.

7.3 In order to show that each fancy term (viewed as 012 e -row) is satisfiable, we introduce some notation. Given a 012 e -row we call a complementary pair $\{x_i, \bar{x}_i\}$ *bad* if it is covered by distinct e -bubbles; otherwise it is *good*. A 012 e -row r' is *purified* if all complementary pairs are good. Such rows r' are nonempty since all e -bubbles can be put to 1 without conflict; indeed, $e \rightarrow 1$ merely forces $2 \rightarrow 0$ for some 2 elsewhere. In order to show that each 012 e -row r contains some purified row r' and whence is satisfiable, it suffices by induction to show that r contains a row r' with one less bad pair $\{x_i, \bar{x}_i\}$. Therefore, say $e_1 e_1 \dots, e_1$ (its first e_1) covers x_i and $e_2 e_2 \dots e_2$ (its first e_2) covers \bar{x}_i . Then r' arises from r by substituting $12 \dots 2$ for $e_1 e_1 \dots e_1$ and $0e_2 \dots e_2$ for $e_2 e_2 \dots e_2$. The case that $e_2 e_2$ is of length two however requires special attention.

	x_1	\bar{x}_1	x_2	\bar{x}_2	x_3	\bar{x}_3	x_4	\bar{x}_4	x_5	\bar{x}_5	x_6	\bar{x}_6	x_7	\bar{x}_7	x_8	\bar{x}_8	x_9	\bar{x}_9
$r =$	e_1	e_4	e_3	e_5	2	e_2	e_6	e_4	e_6	e_1	e_6	e_5	e_4	e_3	e_4	e_1	e_2	2
$r' =$	1	0	e_3	e_5	2	e_2	e_6	e_4	e_6	2	e_6	e_5	e_4	e_3	e_4	2	e_2	2
$r'' =$	1	0	1	0	2	e_2	1	0	2	2	0	1	e_4	2	e_4	2	e_2	2

Table 4: Why 012 e -rows (= fancy terms) are never empty

Then $0e_2 \dots e_2$ boils down to 01. Suppose this 1 occupies x_j and there is a bubble $e_3 e_3$ that covers \bar{x}_j . Then $e_2 e_2 = 01$ forces $e_3 e_3 = 01$, and this pattern may further repeat. However, the number of length two bubbles being finite, one eventually reaches a state where the produced 0 falls upon a 2, or upon a bubble $e_t e_t \dots e_t$ of length ≥ 3 , which then becomes $0e_t \dots e_t$. A concrete example of a 012 e -row r which contains¹⁴ the purified row $r'' \subseteq r' \subseteq r$ is shown in Table 4.

7.3.1 For any purified 012 e -valued row ρ it's easy to calculate $|\text{Mod}(\rho)|$. If say

	x_1	\bar{x}_1	x_2	\bar{x}_2	x_3	\bar{x}_3	x_4	\bar{x}_4	x_5	\bar{x}_5	x_6	\bar{x}_6	x_7	\bar{x}_7	x_8	\bar{x}_8
$\rho =$	e_1	2	2	e_1	2	2	2	2	2	e_2	e_2	2	2	e_2	e_2	2

Table 5: Another example of a purified 012 e -row

then $|\text{Mod}(\rho)| = (2^2 - 1) \cdot (2^4 - 1) \cdot 2^2$. Indeed, for all $2^4 - 1$ legal (i.e. $\neq (0, 0, 0, 0)$) choices of (e_2, e_2, e_2, e_2) the coupled 2's adapt accordingly. Thus if $(e_2, e_2, e_2, e_2) = (\bar{x}_5, x_6, \bar{x}_7, x_8) = (0, 1, 1, 0)$ then $(x_5, \bar{x}_6, x_7, \bar{x}_8) = (1, 0, 0, 1)$. The 2's at x_3 and x_4 are *free* (only coupled to \bar{x}_3, \bar{x}_4)

¹³Actually ESOF implies DNNF even if we dropped (but we won't) the crucial property (b). Similar reasoning shows that DNF implies DNNF.

¹⁴The boldface entries **10** indicate “actively diffused” bad pairs (going from left to right), whereas the other 0 and 1 in r'' are consequences thereof.

and thus can be chosen in 2^2 many ways. Generally, if ρ is a purified 012e-row with e -bubbles of length $\varepsilon_1, \dots, \varepsilon_s$ and t many free 2's then

$$(16) \quad |\rho| = (2^{\varepsilon_1} - 1)(2^{\varepsilon_2} - 1) \dots (2^{\varepsilon_s} - 1) \cdot 2^t$$

7.3.2 In order for formula (16) to be useful we need to show how any 012e-row ρ can be written as a disjoint union of purified rows. An example will do. For ρ as in Table 6 we pick all t bad pairs, here $\{x_1, \bar{x}_1\}, \{x_2, \bar{x}_2\}$, and consider all $2^t = 4$ $\{0, 1\}$ -instantiations ρ_1, \dots, ρ_4 of ρ . Obviously $\rho = \rho_1 \uplus \rho_2 \uplus \rho_3 \uplus \rho_4$, but some ρ_i (despite appearances) may be empty since an e -bubble falls into $\text{zeros}(\rho_i)$; in our case $\rho_2 = \emptyset$ since $e_2 e_2$ falls into $\text{zeros}(\rho_2)$.

	x_1	\bar{x}_1	x_2	\bar{x}_2	x_3	\bar{x}_3	x_4	\bar{x}_4	x_5	\bar{x}_5	x_6	\bar{x}_6
$\rho =$	e_1	e_2	e_2	e_3	e_1	2	e_3	2	2	e_3	e_1	2
$\rho_1 =$	1	0	1	0	2	2	e_3	2	2	e_3	2	2
$\rho_2 =$	1	0	0	1	2	2	2	2	2	2	2	2
$\rho_3 =$	0	1	1	0	e_1	2	e_3	2	2	e_3	e_1	2
$\rho_4 =$	0	1	0	1	e_1	2	2	2	2	2	e_1	2

Table 6: Getting a disjoint union of purified rows

As proven previously at least one of the 2^t many instantiation of ρ will yield a nonempty purified 012e-row ρ_j . Getting these ρ_j 's can be achieved in smarter ways than listing 2^t rows and discarding the bad ones. But we won't dwell on this here.

7.4 The technicalities of the ALLSAT e -algorithm simplify if the CNF for φ features only *positive* literals. To start with, all arising 012e-rows are purified since all components indexed by negative literals carry don't-care 2's. In fact, we may consider in the first place *short* 012e-rows labelled only by the positive literals x_1 to x_w . The satisfiability test becomes straightforward in this scenario: Such a 012e-row r is feasible with respect to a CNF φ with clauses C_1, C_2, \dots, C_h if $C_i \not\subseteq \text{zeros}(r)$ for all $1 \leq i \leq h$. Indeed, if this condition holds, let u be the length w bitstring defined by $\text{zeros}(u) := \text{zeros}(r)$ and $\text{ones}(u) := [w] \setminus \text{zeros}(r)$. Then $u \in r$ and $\varphi(u) = 1$. Thus the weak feasibility Test 1 from Section 3.1 becomes a perfect feasibility test when restricted to positive Boolean functions.

7.4.1 As is well known, models of a positive Boolean function φ and transversals of a hypergraph are “the same thing”. That's why the trimming of the ALLSAT e -algorithm in the case of positive Boolean functions was called *transversal e -algorithm* in [W5]. It is shown in detail in [W5, Sec.2] how the Flag of Papua (see Table 1) is lifted from the 012-level to the 012e-level in the transversal e -algorithm. The author is confident that the Flag of Papua further carries over to the ALLSAT e -algorithm but this has not yet been implemented into Mathematica. We expand further on the past and future of the ALLSAT e -algorithm in Section 9.

7.4.2 We note in passing that for positive Boolean function the following compression advantage of ESOF against ESOP can be proven. There are 3^n many 012-rows of length n , but $\mathcal{Bell}(n+2) - \mathcal{Bell}(n+1)$ many 012e-rows [W5]. Here the *Bell*-number $\mathcal{Bell}(n)$ gives the number of set partitions of $[n]$. For instance $3^{10} = 59049$ whereas $\mathcal{Bell}(12) - \mathcal{Bell}(11) = 4097622$.

7.5 For later purpose consider the task to represent the intersection of two 012e-rows r and

ρ . For simplicity we stick to the case of positive Boolean functions but the arguments readily carry over. One option is to take the row with the fewer and shorter e -bubbles, say it is ρ with e -bubbles of lengths $\varepsilon_1, \dots, \varepsilon_s$, and to expand it into $N := \varepsilon_1 \varepsilon_2 \dots \varepsilon_s$ many 012-rows ρ_i by “multiplying out” s many Flags of Papua. It then follows that

$$r \cap \rho = (r \cap \rho_1) \uplus (r \cap \rho_2) \uplus \dots \uplus (r \cap \rho_N)$$

where each $r \cap \rho_i$ is either empty (when 0’s clash with 1’s) or can readily be written as a 012 e -row (akin to the intersection of two 012-rows in the proof of Corollary 1). The crucial words here are “fewer and shorter”. Thus if ρ has e -bubbles of lengths $\varepsilon_1 = \varepsilon_2 = \dots = \varepsilon_6 = 10$ then it induces $N = 10^6$ many 012-rows ρ_i ! In such a case one is better off picking the row with the fewer e -bubbles, say again ρ , and impose them on the other row by virtue of the transversal e -algorithm. For illustration see Table 7 where the five 012 e -rows below r have a disjoint union that equals $r \cap \rho$. In particular $|r \cap \rho| = 63 + 12 + 6 + 42 + 18 = 141$. If only the cardinality of $r \cap \rho$ is required (as in 9.3.4) one may be better off using inclusion-exclusion. Namely, consider the property p_1 of a bitstring $u \in r$ to satisfy $eeee$ (thus $|\text{ones}(u) \cap \{1, 2, 3, 4\}| \geq 1$). Similarly p_2 holds if $\bar{e}\bar{e}\bar{e}\bar{e}$ is satisfied. If say $N(\bar{p}_1)$ is defined as the number of $u \in r$ *not* satisfying p_1 we get

$$\begin{aligned} |r \cap \rho| &= |r| - N(\bar{p}_1) - N(\bar{p}_2) + N(\bar{p}_1 \bar{p}_2) \\ &= 3 \cdot 7 \cdot 7 - |(0, 0, 0, 0, e_1, e_1, 1, 1)| - |(1, 1, e_3 e_3, 0, 0, 0, 0)| + 0 \\ &= 147 - 3 - 3 + 0 = 141 \end{aligned}$$

which matches the number obtained above.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
$\rho =$	e	e	e	e	\bar{e}	\bar{e}	\bar{e}	\bar{e}	
$r =$	e_1	e_2	e_3	e_3	e_1	e_1	e_3	e_2	
	1	e_2	e_3	e_3	e_4	e_4	e_3	e_2	63
	1	e_2	2	2	0	0	1	e_2	12
	1	2	e_3	e_3	0	0	0	1	6
	0	1	e_3	e_3	e_1	e_1	e_3	2	42
	0	0	e_3	e_3	e_1	e_1	2	1	18

Table 7: The intersection of two 012 e -rows can be calculated by the transversal e -algorithm

8 Numerical experiments

In brief, 8.1 compares the variable-wise ALLSAT 012-algorithm of Section 5.1 with the clause-wise ALLSAT 012-algorithm of Section 6. In 8.2 the clause-wise ALLSAT 012-algorithm is pitted against the (clause-wise) ALLSAT e -algorithm of Section 7. In 8.3 the clause-wise ALLSAT 012-algorithm is pitted against BDD’s. Finally 8.4 adds weight functions to the picture.

8.1 We will pit the *variable*-wise ALLSAT 012-algorithm against the *clause*-wise ALLSAT 012-algorithm on Boolean functions $\varphi : \{0, 1\}^w \rightarrow \{0, 1\}$ with h random clauses, each of cardinal-

ity λ for simplicity. For each triplet (w, h, λ) we only¹⁵ produced few φ 's. For φ we record $|\text{Mod}(\varphi)|$ and the times in seconds¹⁶ it took to enumerate $\text{Mod}(\varphi)$ (with disjoint 012-rows) using the variable-wise, respectively clause-wise ALLSAT 012-algorithm. Furthermore, for both algorithms $|\gamma|$ gives the average number of 2's per final row. Thus the higher γ the better the compression¹⁷ We wrote ≈ 0 for probabilities $< 10^{-6}$. The (25, 50, 10) and (100, 5, 30) and (500, 3300, 4) instances show that variable-wise badly trails behind clause-wise when $|\text{Mod}(\varphi)|$ is large. Here "68/sec" means that *during the first hour* an average of 68 models per second were produced but the algorithm could not finish. Similarly for 0.16/sec. When $|\text{Mod}(\varphi)|$ is small then both algorithms play on level ground because the number R of final 012-rows necessarily is $\leq |\text{Mod}(\varphi)|$. As h gets very large the variable-wise approach starts to win out as seen in the (25, 50000, 12) instance.

w	h	λ	$ \text{Mod}(\varphi) $	variable-wise			clause-wise		
				γ	Time	prob.	γ	Time	prob.
25	50	10	31'949'980	3.9	3596	0.866	9.6	36	0.495
25	9000	10	5251	0	6580	0.0002	0	6385	0.002
25	15000	10	12	0	53	≈ 0	0	65	≈ 0
25	50 000	12	140	0	2516	≈ 0	0	3599	≈ 0
100	5	30	$\approx 2^{100}$?	$> 10^{18}$ yrs	?	87.6	8	0.674
100	900	4	1300	0	498	≈ 0	0.7	292	≈ 0
500	3300	4	?	0	0.16/sec	≈ 0	7.8	68/sec	≈ 0

Table 8: Variable-wise versus clause-wise branching on the 012-level

8.2 Already when comparing the (cut short) Table 2 with Table 3 it becomes plausible that introducing the e -symbol can speed up further the clause-wise approach. While the clause-wise ALLSAT 012-algorithm has been programmed with MATHEMATICA, recall from 7.4 that this is pending for the ALLSAT e -algorithm. However, the two algorithms *can* be compared numerically for the special case of positive Boolean functions. Specifically, as pointed out in 7.4, the ALLSAT e -algorithm, fed with a positive Boolean function φ , behaves exactly as the transversal e -algorithm when fed with φ .

Let us hence compare the (clause-wise) transversal e -algorithm with the (clause-wise) transversal 012-algorithm. Without further mention, all φ 's in 8.2 are positive. For starters, when φ is given by h mutually *disjoint* clauses of lengths $\varepsilon_1, \dots, \varepsilon_h$ the difference in compression is dramatic: While just *one* 012e-row suffices to represent $\text{Mod}(\varphi)$, it takes $\varepsilon_1 \varepsilon_2 \dots \varepsilon_h$ many 012-rows to achieve the same thing. Also for $\varphi(x_1, \dots, x_w)$ with h *random* clauses, each of length λ (for simplicity), the numerical evidence in favor of the transversal e -algorithm is compelling, as shown in Table 9. Here T_e and T_{012} are the times in seconds needed by the transversal e -, respectively 012-algorithm. Similarly R_e and R_{012} are the respective numbers of final 012e-rows and 012-rows.

¹⁵It turns out that random φ 's sharing the same parameters (w, h, λ) behave very much alike.

¹⁶Except for one instance where (extrapolated) *years* are more telling.

¹⁷For both algorithms we also record $\text{prob} = \text{prob}(w, \gamma, h, \lambda)$ as defined in Section 3.3. Hence prob is the probability that a *random* 012-row r (independent of any algorithm) with parameters (w, γ, h, λ) also happens to satisfy $r \subseteq \text{Mod}(\varphi)$. Though interesting, this isn't relevant in the present context.

(w, h, λ)	$ \text{Mod}(\varphi) $	R_e	T_e	R_{012}	T_{012}
200 10 150	$\approx 2^{200}$	802	0.2	8×10^5	61
60, 40, 30	$\approx 10^{18}$	134392	56	9×10^6	563
60, 25, 7	$\approx 10^{18}$	841531	292	6×10^9	10^6
20 10 4	650 024	37	0.03	218	0.05
20 50 4	243 632	2036	1.4	11 669	1.8
20 100 4	129 206	4961	5.7	15909	3.8
20 3000 4	4717	2365	132	3220	18
20 3000 15	1039831	3972	177	16879	6.3
40 50000 4	107957	-	-	87833	10662

Table 9: *Transversal e-algorithm against transversal 012-algorithm (for positive functions)*

For the (200, 10, 150), the (60, 40, 30) and the (60, 25, 7) instances one has $R_e \ll R_{012}$ and whence $T_e \ll T_{012}$. In fact for (60, 25, 7) the transversal 012-algorithm was stopped after 21 hours and the values of T_{012} and R_{012} are only extrapolated. Letting $h = 10, 50, 100, 3000$ in the $(20, h, 4)$ instances one still has $R_e < R_{012}$ but these gaps get proportionally smaller and are eventually time-wise more than compensated by the simpler row-splitting mechanism on the 012-level. Even more so in the (20, 3000, 15) and (40, 50000, 4) instances.

8.2.1 If condition (7a) is not maintained by the row-splitting algorithm then some of the rows r in the working stack will be infeasible, and they may trigger infeasible sons. However, eventually infeasibility will be detected. To fix ideas, if say the clause $x_2 \vee x_4 \vee x_7$ is to be imposed on the top row r , and $\{2, 4, 7\} \subseteq \text{zeros}(r)$, then r is detected as infeasible and must be deleted. Such a deletion we call *harmful* (as opposed to the deletion of a feasible row when it is replaced by its sons). Recall that Test 1 in Section 3 becomes a *perfect* feasibility test for positive Boolean functions. However, the extra time to run the test (and hence avoid harmful deletions of rows) doesn't always pay off. In the instances above it only paid off for $h \geq 3000$. Thus if we don't use Test 1 for the (20, 3000, 4) instance then the transversal e -algorithm suffers 11384 harmful row deletions and the time jumps from 132 to 243 seconds. For $h < 3000$ we were better off with accepting harmful deletions.

8.2.2 One may think that for small values w checking all $u \in \{0, 1\}^w$ *individually* may be faster than imposing a large number h of constraints. Thus we implemented this “naive way” to scan¹⁸ the modest-size powerset $\{0, 1\}^{20}$ and found that the (20, 3000, 15)-instance in Table 9 took a hefty 12830 sec. The problem is that a small powerset doesn't help because, in view of $k = 15$, most $u \in \{0, 1\}^{20}$ suffer all 3000 intersection tests ($\text{ones}[u] \cap \text{clause}[i] \stackrel{?}{=} \emptyset$). We hence also ran the (20, 3000, 4)-instance in the naive way. Indeed the time boiled down to 344 sec since most $u \in \{0, 1\}^{20}$ dropped out after few intersection tests. The joy is short: Increasing w to 40 (as in the (40, 50000, 4)-instance) renders a mere *scanning* of $\{0, 1\}^{40}$ (without extra fuzz) out of question.

8.3 Now we pit the clause-wise ALLSAT 012-algorithm against the MATHEMATICA command `SatisfiabilityCount` $[\varphi]$. The latter works by constructing the BDD of φ , which then yields $|\text{Mod}(\varphi)|$ at once (see 2.2). Hence timing `SatisfiabilityCount` in effect means timing the construction of a BDD. Unfortunately this is only half the task we wish to time. Since the

¹⁸This is easily established with the Mathematica command `Subsets[...]`.

underlying BDD seems to be inaccessible to the user, we cannot¹⁹ assess getting an ESOP from the BDD; neither the time it takes nor the compression (= number of 012-rows) achieved. This constitutes on “unfair advantage” in the timing of **SatisfiabilityCount**. The second advantage is the fact that it is a “hardwired” MATHEMATICA command whereas the clause-wise ALLSAT 012-algorithm is written in high-level MATHEMATICA code. As a perfect feasibility test we use the hardwired MATHEMATICA command **SatisfiabilityInstances**. It is based²⁰ on a search tree and either offers a model for any Boolean function φ or it returns the empty set, in which case φ is provably unsatisfiable.

In Table 10 on the left T_{SC} is the time for **SatisfiabilityCount**, T_{012} the time for the clause-wise ALLSAT 012-algorithm, and R_{012} the number of final 012-rows it produces. If we compare the $(60, 10, k)$ instances for $k \in \{3, 7\}$, **SatisfiabilityCount** is way ahead (keeping in mind its unfair advantages). The time T_{012} is essentially proportional to the number R_{012} of final rows, which in turn depends on how often the “Flag of Papua gets raised” (Section 6.1). If we push $h = 10$ to 50 then $T_{SC} = 78$ sec, and for $h = 80$ we stopped **SatisfiabilityCount** after 50 fruitless hours. Admittedly, also T_{012} and R_{012} are astronomic for $h = 50, 80, \dots$ but, different from **SatisfiabilityCount**, not forever! Namely, if say $h = 4600$ then this sheer number of clauses allows only for 143 models which the ALLSAT 012-algorithm found in 6191 seconds. Observe that **SatisfiabilityInstances** took 83 sec to find just *one* model, while the ALLSAT 012-algorithm required $6191/143 \approx 43$ sec per model on average, despite the fact that **SatisfiabilityInstances** itself is an essential ingredient of it.

8.3.1 A few words about the weak feasibility tests in Section 3 are in order. While Test 1 becomes a perfect feasibility test for positive Boolean functions, it performs poorly for arbitrary Boolean functions. Interestingly the *conjunction* (Test 1 + Test 2) yields a decent weak feasibility test, i.e. with few harmful deletions of intermediate 012-rows. Time-wise however (Test 1 + Test 2) couldn’t compete with **SatisfiabilityInstances**. Thus the former needed 4755 sec to find the two models of some $(50, 490, 4)$ instance (not in Table 10) and suffered 431059 harmful deletions, while the latter did the job in 0.5 sec. Whether this state of affairs changes when other weak feasibility tests are added, or when the hardwire-advantage of **SatisfiabilityInstances** is taken into account, remains to be seen. As to **SatisfiabilityCount**, we aborted it after 16 hours.

8.4 Recall from 6.1 that the domain of our Boolean functions $\varphi(x_1, \dots, x_w)$ is $\{x_1\bar{x}_1, \dots, x_w, \bar{x}_w\}$. Consider now weight functions $f : \{0, 1\}^{2w} \rightarrow \mathbb{Z}_+$ induced²¹ by random functions $[2w] \rightarrow [20]$. Using BDD’s the *whole* BDD needs to be available in order to sieve all models of small weight (say $\leq b$). This is not the case for the clause-wise ALLSAT 012-algorithm. Namely, along with the perfect feasibility test applied to an intermediate row r one can check fast whether r contains any bitstrings (thus possibly models) of weight $\leq b$. If no, then r is deleted. Of course this idea beats first producing all models and then throwing most of them away (which essentially the BDD approach is doomed to do). For each instance (w, h, k) binary search quickly²² yields values $b_0 \in \mathbb{Z}_+$ which are large enough to trigger a nonempty set $\text{Mod}(\varphi, f, b_0)$ of models of weight $\leq b_0$, yet small enough to keep $|\text{Mod}(\varphi, f, b_0)|$ at bay.

¹⁹Mending this state of affairs is a major task of a planned follow-up paper.

²⁰Thus not on a BDD, and also not on the popular Davis-Putman-Logemann-Loveland algorithm (DPLL).

²¹The number 20 in the function $g : [2w] \rightarrow [20]$ is arbitrary. If say $w = 3$ and $(x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3) = (1, 0, 1, 0, 0, 1)$ then $f((1, 0, 1, 0, 0, 1)) := g(1) + g(3) + g(6)$.

²²We didn’t time this preprocessing part.

To fix ideas, let us return to the $(60, 50, 7)$ instance where T_{SC} is a hefty 78 sec and both T_{021} and R_{012} are astronomic. Here the time to enumerate only the models of weight $\leq b_0$ was a mere $T'_{012} = 0.4$ sec (see Table 10 on the right). For this particular b_0 there were exactly $R'_{012} = 8$ final 012-rows which contained at least one small weight model. In the process $del = 228$ times an intermediate 012-row r was deleted because all $u \in r$ had weight $> b_0$. For $h = 80$ all of T_{SC}, T_{012}, R_{012} are astronomic yet for suitable small b_0 one gets $T'_{012} = 0.2$ and $R'_{012} = 4$. Raising b_0 a bit yields $T'_{012} = 81$ and $R'_{012} = 4446$. When R_{012} is small already, as in the $(60, 4600, 7)$ instance, then T'_{012} can't be pushed much below T_{012} .

(w, h, k)	$ \text{Mod}(\varphi) $	T_{SC}	T_{012}	R_{012}	T'_{012}	R'_{012}	del
60 10 3	$\approx 3 \times 10^{17}$	0	7	6318	0.1	10	63
60 10 7	$\approx 10^{18}$	0	69	71470	0.7	52	313
60 50 7	$\approx 8 \times 10^{17}$	78	-	-	0.4	8	228
60 80 7	$\approx 6 \times 10^{17}$	-	-	-	0.2	4	156
60 80 7	$\approx 6 \times 10^{17}$	-	-	-	81	4446	32953
60 4600 7	143	-	6191	139	6154	2	122

Table 10: SatisfiabilityCount (= BDD) against the clause-wise ALLSAT 012-algorithm

9 History and envisaged future

History and intended future of the ALLSAT e -algorithm will be addressed in 9.1 to 9.2. In particular, Subsection 9.2 dwells on BDD's and draws part of its optimism from the computer experiments in Section 8. Subsection 9.3 is still about history and future but zooms away from the ALLSAT e -algorithm to *specific* types of ALLSAT problems susceptible to wildcards.

9.1 In 2005 I gave a talk about the transversal e -algorithm (see 7.4.1). This inspired Gideon Redelinghuys and Jaco Geldenhuys [RG] to carry over the e -framework from positive to *arbitrary* Boolean CNF's, in order to determine their satisfiability; they called their method SATE-algorithm. In the last 10 years I more or less forgot SATE and turned to clause-wise branching in more *specific* scenarios as will be surveyed in 9.3. Yet I recently returned to the SATE algorithm and now regard the extension of the e -symbolism from $\{x_1, \dots, x_w\}$ to $\{x_1, \bar{x}_1, \dots, x_w, \bar{x}_w\}$ as its crucial idea. However, the attempt to tackle Chaff, one of the leading SAT-solvers [MMZZM], is misguided. Clause-wise branching should not be abused to challenge tailor-made satisfiability tests. Rather clause-wise branching fits ALLSAT like a glove: Once all clauses are imposed on a multivalued²³ row, it is automatically final and may pack a great many models. What is more, a node in the search-tree of clause-wise (as opposed to variable-wise) branching can have more than two sons. In many circumstances (Table 8) this makes branching more efficient.

Not knowing any technical details²⁴ of the C -implementation of the SATE algorithm in [RG], I embarked to reprogram it from scratch in high-level Mathematica code. After some deliberation I settled for the 012-level, thus programming the clause-wise ALLSAT 012-algorithm (Section 6). The comparison of the transversal 012-algorithm with the transversal e -algorithm in 7.2, as

²³Here "multivalued" means that apart from 0, 1 and the don't-cares 2 one has further wildcards such as (e, e, \dots, e) and others that were useful.

²⁴Whatever they are, a glimpse at [RG] shows that e.g. the issue of purified rows has been glossed over.

well as the comparison of Tables 2 and 3, strongly indicate that the ALLSAT e -algorithm, once programmed²⁵, will exhibit a further leap in compression. Predictably ALLSAT-algorithms that output their models one-by-one will trail when the number of models gets large.

9.2 Probably the main competitor of the ALLSAT e -algorithm is the BDD framework since, as seen in Section 2, a BDD for φ allows an enumeration of $\text{Mod}(\varphi)$ by 012-rows (ESOP). But there are several issues that need to be investigated; here in 9.2 we only glimpse at them.

9.2.1 For starters, there is the conversion time from CNF to BDD. Then there is the often denied fact that the average size of the BDD of a Boolean function $\varphi(x_1, \dots, x_n)$ is about $2^n/n$, thus a hefty $\frac{1}{n}$ times the length of the full truth table of φ . True, changing the variable order²⁶ often shrinks the BDD, but that costs time. On an aesthetic level, the construction algorithms for BDD's are awkward (even in Knuth's otherwise lovely introduction [K] to BDD's). In contrast the ALLSAT e -algorithm can be grasped in a more visual manner (Table 3).

9.2.2 Assume we *have* (in whatever way obtained) a BDD and also an ESOF of some Boolean function φ . Using the BDD calculating $|\text{Mod}(\varphi)|$ is fast, and enumerating $\text{Mod}(\varphi)$ is straightforward. Straightforward it may be, but if the index gaps (2.3) are small, many 012-rows may be bitstrings. Concerning the ESOF, after “purification” as in 7.3.2, we have a representation of $\text{Mod}(\varphi)$ as disjoint union of purified 012e-rows, which have more compression potential than 012-rows (see 7.4.2).

9.2.3 What about fixed cardinality models? Calculating $|\text{Mod}(\varphi, k)|$ from a purified ESOF of φ is readily reduced to calculating the coefficient at x^k of some associated polynomial $p(x)$, exactly as in [W5. p.124]. Enumerating $\text{Mod}(\varphi, k)$ from a purified ESOF can be done with wildcards in a manner similar to [W11]. As to BDDs, calculating $|\text{Mod}(\varphi, k)|$ from a BDD of φ is little known: A nice method of Knuth [K, Exercise 25] is reviewed (with trimmed notation) in [W10]. The enumeration of $\text{Mod}(\varphi, k)$ from a BDD of φ , as handled in [W10], seems to be new. As opposed to ESOF it can actually be done in polynomial total time (and again with wildcards). Polynomial total time or not, how ESOF compares to BDD in practise remains to be seen.

9.2.4 One major benefit of BDDs is *equivalence* testing: Given Boolean formulas φ and ψ it holds that $\varphi \leftrightarrow \psi$ if and only if the corresponding BDD's are isomorphic. This fails for ESOF since φ has many different ESOFs. However things aren't too bad. If φ and ψ are in purified ESOF format then $N(\varphi) := |\text{Mod}(\varphi)|$ and $N(\psi) := |\text{Mod}(\psi)|$ are readily computed, and if $N(\varphi) \neq N(\psi)$ then $\varphi \not\leftrightarrow \psi$. Conversely, if $N(\varphi) = N(\psi)$ then chances are high that $\varphi \leftrightarrow \psi$. Even more so if $|\text{Mod}(\varphi, k)| = |\text{Mod}(\psi, k)|$ for all k (see 9.2.3).

In order to sketch a waterproof equivalence test, let $\text{Mod}(\varphi)$ and $\text{Mod}(\psi)$ be given as purified ESOFs, say $r_1 \uplus \dots \uplus r_s$ and $r'_1 \uplus \dots \uplus r'_t$ respectively. Suppose we checked that $N(\varphi) = N(\psi)$ and we managed to prove for all $1 \leq i \leq s$ that

$$(17) \quad \text{Card}(r_i) = \sum_{j=1}^t \text{Card}(r_i \cap r'_j).$$

²⁵I leave that noble task to others. It would be nice if Chaff was used as satisfiability subroutine. If the ALLSAT e -algorithm gets coded in Mathematica, `SatisfiabilityInstances` is a decent substitute.

²⁶The order in which one feeds the clauses to the ALLSAT e -algorithm similarly influences the size of the resulting ESOF but this has not been researched yet.

Then (17) implies that all $r_i \subseteq r'_1 \uplus \dots \uplus r'_t$, whence $r_1 \uplus \dots \uplus r_s \subseteq r'_1 \uplus \dots \uplus r'_t$, whence $\text{Mod}(\varphi) = \text{Mod}(\psi)$ in view of $N(\varphi) = N(\psi)$. Conversely, if (17) fails for some $i \in [s]$ then $\text{Mod}(\varphi) \neq \text{Mod}(\psi)$. As to calculating $\text{Card}(r_i \cap r'_j)$, we mention that the inclusion-exclusion method of 7.5 can be sped up (work in progress). Even so, the ensuing equivalence test is wanting when compared to the one provided by a BDD.

This suggests a hybrid method for checking whether $\varphi \leftrightarrow \psi$: If $N(\varphi) \neq N(\psi)$ (as swiftly calculated with the ALLSAT e -algorithm), then $\varphi \not\leftrightarrow \psi$. Otherwise invest calculating BDD's of φ and ψ to settle the question. Of course, if the likelihood for $\varphi \leftrightarrow \psi$ somehow is high *beforehand* then use BDD's right away.

9.3 In the last ten years the author found compressed representations of specific types of set systems by employing the don't-care “2” and various wildcards apart from $ee \dots e$. Let us briefly take stock. Article [W2] enumerates the model set of a Horn formula φ as a disjoint union of $012n$ -rows. Here the wildcard $nn \dots n$ means (dually to $ee \dots e$) “at least one 0 here”. The method of [W2] can be fine-tuned for three special types of Horn formula. First, the noncover n -algorithm applies to negative Boolean functions and was successfully applied to stack filters, i.e. tools used in nonlinear digital filtering [W3]. Second, *pure* Horn functions, aka *implications* $A \rightarrow B$, enjoy many applications in Data Mining and elsewhere [W7], [W8]. Third, particularly pleasant implications are the ones with singleton premises $A = \{a\}$. The corresponding (a, b) -algorithm calculates all order ideals of a poset in a compressed fashion [W4]. Note that both for stack filters and for Coupon Collecting [W6] it is essential not merely to calculate $|\text{Mod}(\varphi)|$ but also $|\text{Mod}(\varphi, k)|$ for *all* $k \in [w]$. Article [W1] was the wildcard pioneer. While the wildcards are used *during* the algorithms, all final rows are bitstrings. This is not due to deficient programming but forced by the inherent structure of the models²⁷. Hence traditional one-by-one algorithms become competitive, and actually proved superior for chordless cycles. With hindsight it is clear that the employed wildcards could be used more efficiently (work in progress) if the target isn't exhaustive enumeration of Hamiltonian cycles, but rather the confirmation of suspected *non*-Hamiltonian graphs. The compression of $\text{Mod}(\varphi)$ for 2-CNF's [W9] also is an outsider in that it is achieved using *variable*-wise branching and nothing fancier than don't-care 2's. The fact that all mentioned $\text{Mod}(\varphi)$ (except [W1]) can be compressed in polynomial total time is due to the fact that the corresponding formulas φ allow for a polynomial-time satisfiability test.

9.3.1 The present article is conceived as a hinge between past and future. We just reviewed the past, and Theorem 2 is intended to be a hat for several future results. Likely their inner and (more so) interdependent structure will be, with the benefit of hindsight, a bit better organised than for [W1] to [W6]. Specifically the present article is Part 1 in a planned series “ALLSAT compressed with wildcards”. The present Part 1 tackled *arbitrary* Boolean functions, mainly CNF's. The more thought-through next topics in the planned mini-series concern again *specific* Boolean functions: Enumerating all faces of a simplicial complex (given by its facets or otherwise); all shellings of a simplicial complex; all connected subgraphs of a graph; all anticliques of a graph; all k -models of a BDD, all closed sets of a convex geometry. The order of publication, or in fact publication itself, cannot be predicted at this stage.

²⁷Speaking of “models” is inaccurate here since Hamiltonian cycles in a graph cannot be viewed as models of an underlying Boolean function. In fact constraint-wise instead of clause-wise branching better fits the bill. Unfortunately “constraint-wise” may evoke unwanted proximity to Constraint Programming. The precise relation between the two remains to be unravelled.

References

- [D] A. Darwiche, New advances in compiling CNF to decomposable negation normal form, Proc. of the European Conference on Artificial Intelligence 2004.
- [FG] J. Flum, M. Grohe, Parametrized Complexity Theory, Springer 2006.
- [K] D. Knuth, the Art of Computer Programming, Volume 4 (Preprint), Section 7.14: Binary decision diagrams, Addison-Wesley 2008.
- [M] A. Marino, Analysis and enumeration. Algorithms for biological graphs. This price-winning Thesis appeared as Vol.6 of Atlantis Studies in Computing (2015).
- [MMZZM] M.W. Moskewicz,, C.F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT Solver, 39th Design Automaton Conference, Las Vegas ACM 2001.
- [R] M. Rospocher, On the complexity of enumerating certificates of NP-problems, PhD Thesis 2006, University of Trento, Italy.
- [RG] G. Redelinghuys, J. Geldenhuys, A new and complete refinement-based approach to Boolean satisfiability, unpublished manuscript 2005.
- [SSK] P. Suter, R. Steiger, V. Kuncak, Sets with cardinality constraints in satisfiability modulo theories, LNCS 6538, 2011, 403-418.
- [V] L.G. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (1979) 410-421.
- [Wa] M. Wahlström, Exact algorithms for finding minimum transversals in rank 3 hypergraphs, Journal of Algorithms 51 (2004) 107-121.
- [Was] K. Wasa, Enumeration of Enumeration Algorithms, arXiv: 1605.05102v1, 2016.
- [W1] M. Wild, Generating all cycles, chordless cycles and Hamiltonian cycles with the principle of exclusion. J. Discrete Algorithms 6 (2008), no.1 93-102.
- [W2] M. Wild, Compactly generating all satisfying truth assignments of a Horn formula, J. Satisf. Boolean Model. Comput. 8 (2012), no.1-2, 63-82.
- [W3] M. Wild, Computing the output distribution and selection probabilities of a stack filter from the DNF of its positive Boolean function. J. Math. Imaging Vision 46 (2013), no.1, 66-73.
- [W4] M. Wild, Output-polynomial enumeration of all fixed-cardinality ideals of a poset, respectively all fixed-cardinality subtrees of a tree. Order 31 (2014), no.1, 121-135.
- [W5] M. Wild, Counting or producing all fixed cardinality transversals. Algorithmica 69 (2014), no.1, 117-129.
- [W6] M. Wild, S. Janson, S. Wagner, D. Laurie, Coupon collecting and transversals of hypergraphs. Discrete Math. Theor. Comput. Sci. 15 (2013), no.2, 259-270.
- [W7] M. Wild, The joy of implications, aka pure Horn formulas: Mainly a survey. Theoretical Computer Science 658 (2017) 264-292.

- [W8] M. Wild, Compressed representation of Learning Spaces, to appear in J. Math. Psychology.
- [W9] M. Wild, Revisiting the enumeration of all models of a Boolean 2-CNF, in the arXiv.
- [W10] M. Wild, ALLSAT compressed with wildcards. Part 2: All weight k models of a BDD. In preparation.
- [W11] M. Wild, ALLSAT compressed with wildcards Part 3: How to get partitionings and face-numbers of simplicial complexes. In preparation.